Parallel computation of the rank of large sparse matrices from algebraic K-theory

Jean-Guillaume Dumas

Laboratoire Jean Kuntzmann, UMR CNRS 5224
Université Joseph Fourier,
B.P. 53 X, 38041 Grenoble, France.

Jean-Guillaume.Dumas@imag.fr

Philippe Elbaz-Vincent

Institut de Mathématiques et de Modélisation de Montpellier,
UMR CNRS 5149
Université Montpellier II, CC051, Place E. Bataillon.
34095 Montpellier cedex 5, FRANCE.

pev@math.univ-montp2.fr

Pascal Giorgi

Laboratoire LP2A, Université de Perpignan Via Domitia.
52, avenue Paul Alduy 66860 Perpignan France.

pascal.giorgi@univ-perp.fr

Anna Urbańska

Laboratoire Jean Kuntzmann, UMR CNRS 5224
Université Joseph Fourier,
B.P. 53 X, 38041 Grenoble, France.

Anna.Urbanska@imag.fr

ABSTRACT

This paper deals with the computation of the rank and some integer Smith forms of a series of sparse matrices arising in algebraic K-theory. The number of non zero entries in the considered matrices ranges from 8 to 37 millions. The largest rank computation took more than 35 days on 50 processors. We report on the actual algorithms we used to build the matrices, their link to the motivic cohomology and the linear algebra and parallelizations required to perform such huge computations. In particular, these results are part of the first computation of the cohomology of the linear group $GL_7(\mathbb{Z})$.

1. INTRODUCTION

1.1 Motivation from K-theory

Numerous problems in modern number theory could be solved or at least better understood, if we have a good knowledge on the algebraic K-theory (or motivic cohomology) of integers of number fields or the cohomology of arithmetic groups (i.e. subgroups of finite index of $GL_N(\mathbb{Z})$). As a short list, we could mention:

- modular forms and special values of L functions,
- Iwasawa theory and understanding of the "cyclotomy",
- Galois representations (or automorphic representations).

Let us explain first what is algebraic K-theory: to a commutative ring R we can associate (functorially) an infinite family of abelian groups $K_n(R)$ which encodes a huge amount of information on its arithmetic, geometric and algebraic structures. These groups extend some classical notions and give higher dimensional analogues of some well known results. For instance if R is a ring of integers or a polynomial ring over a field, we have

- $K_0(R)$ is the classical Grothendieck group (classifying finitely generated $R-$modules),
- $K_1(R)$ is the group of invertibles of R,
- $K_2(R)$ classifies the universal extensions of $SL(R)$ and is related to the Brauer group in the case of a field.

We can give a general abstract definition of K_n for $n > 0$,

$$K_n(R) = \pi_n(BGL(R)^+)\,,$$

where $BGL(R)^+$ is the Quillen $+$-construction applied to the classifying space $BGL(R)$ and π_n denotes the nth homotopy group [23]. We also can see the K-groups as a way to understand $GL(R)$. For instance, $K_1(R)$ is isomorphic to $GL(R)$ modulo elementary relations. For a more detailed background on K-theory and its applications see [23].

Fact: these groups are hard to compute.

1.1.1 The Vandiver conjecture as an illustration

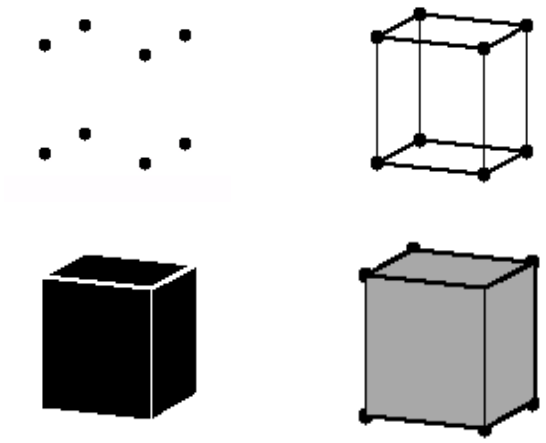
The Vandiver conjecture plays an important role in the understanding of the "cyclotomy" in number theory. Its statement is as follows;

Conjecture of Vandiver: Let p be an odd prime, $\zeta_p = e^{2\pi i/p}$, C the p-Sylow subgroup of the class group of $\mathbb{Q}(\zeta_p)$ and C^+ the subgroup fixed by the complex conjugation. The

Vandiver conjecture is the statement that $C^+ = 0$.
To illustrate its interplay with K-theory, we have
Fact (Kurihara, 1992)[22]: If $K_{4n}(\mathbb{Z}) = 0$ for all $n > 0$, then the conjecture of Vandiver is true.
Some partial results on this conjecture and its connection with the cohomology of $SL_N(\mathbb{Z})$ are given in [25].
General problem: Find explicit methods for computing (co)homologies of arithmetic groups and the K-theory of number fields (or their ring of integers).
Our first task will be the computation of the (co)homologies of linear groups (mainly $GL_N(\mathbb{Z})$ and $SL_N(\mathbb{Z})$). We can show [15, 13, 14] that the computation of those groups is the key point for computing K-groups with our method. We will begin to recall some facts from topology.

1.1.2 Topological Excursion: Cellular complexes, or how to simply describe the "combinatorial" structure of a topological space

The notion of cellular complex is a generalization of a graph to several dimensions. We call *$n-$cell* a topological space homeomorphic to the open unit ball of $\mathbb{R}^n$ and such that its closure is homeomorphic to the closed unit ball. A cellular complex (or cell complex or also cellular decomposition or cellular space) is a family of sets X^n (with $n \in \mathbb{N}$), such that each X^n is a collection (eventually infinite) of $n-$cells. Usually we work with cell complexes with a finite number of cells.
A classical result [26] shows that *any (reasonable) topological space can be approximated by such cell complexes.*



A cellular decomposition of the cube

1.1.3 Computation of the homology of a cell complex

To a cell complex, we can associate a family C_n ($n \in \mathbb{N}$) of free $\mathbb{Z}-$modules and a family $d_n : C_n \to C_{n-1}$ of linear maps.
The module C_n is the free module with basis the $n-$cells (modulo a choice of orientation).
If the complex is finite, then all the modules are of finite rank and we will denote by $\{b_\lambda^n\}_{\lambda \in \Lambda_n}$ a basis of C_n, Λ_n being an index set for the $n-$cells. Then the map d_n is defined by

$$d_n(b_\lambda^n) = \sum_\mu [b_\lambda^n : b_\mu^{n-1}] b_\mu^{n-1} ,$$

and the integer number $[b_\lambda^n : b_\mu^{n-1}]$ is called *the incidence number* of the cell e_μ^{n-1} inside the cell e_λ^n. The relation $d_n \circ d_{n+1} = 0$ (i.e., formally $d^2 = 0$) should hold for any n.
If the complex is *regular* (i.e., always at most one cell of dimension $n+1$ between two cells of dimension n), then we can build the incidences inductively starting from the $0-$cells up to the maximal cells using the $d^2 = 0$ condition and moreover the incidence numbers will be 0, or ± 1.
The nth homology group of the complex is defined as the quotient of $\mathrm{Ker}(d_n)$ by $\mathrm{Im}(d_{n+1})$. This construction is functorial (in the category of cell complexes). As a consequence, *we can determine the homology groups effectively by computing the Smith form of the integral matrices of the d_n (relatively to the fixed basis).* Notice that the Smith form gives both the rank of the free part and the explicit description of the torsion part. In case the computation of the torsion is unnecessary (or to difficult to achieve), we can tensorize by $- \otimes_{\mathbb{Z}} \mathbb{Q}$ and the homology groups become $\mathbb{Q}$-vector spaces with their dimensions given by the ranks of the matrices of the differentials.

In general, the matrices of the differentials can be very large even for a relatively simple cell complex. However, they are also very sparse and we may look for some other favorable properties which would enable the computation despite the size of the problem.

1.1.4 How to use such settings for the computation of linear groups ?

If G is a group acting on a cell space X (i.e., G sends $n-$cells to $n-$cells), then, under some technical assumptions on X and on the action, we can show [4] that roughly *computing the homology of G (as group homology) is the same as computing the homology of the cell space X/G.* Hence, if X/G can be calculated effectively, we can compute explicitly its homology, and from this the homology of G (similarly for the cohomology). Notice that in general the space X/G will not be regular anymore. The main difficulty is to find a cell space X such that X/G will be effective. We will discuss in section 2.1 how we can construct such cellular space for linear groups.

1.2 Parallelism motivations

This first idea to deal with very large sparse matrices is to use them as blackboxes, i.e. only using the matrix-vector product. This will let the matrix remain sparse all along the algorithm where Gaussian elimination for instance would fill it up. To compute the rank, the fastest black box algorithm is Wiedemann's as shown in [10]. This algorithm computes a sequence of scalars of the form $u^t A^i v$ (u and v are vectors) with i matrix vector products and dot products. It has been shown to successfully deal with large sparse matrix problems from homology, see e.g. [9]. Nevertheless, when matrices are very large (e.g millions on non-zero entries) computations would require months or years. This is due to the low practical efficiency of the computation of a sparse matrix-vector product. For instance, in our case of homology computation, one would need 300 days of CPU to compute the sequence involving matrix of $GL_7(\mathbb{Z})$ with $n = 19$ (GL7d19 matrix). To achieve computations of many homologies in a realistic time we then need to parallelize the computation of the sequence. Then the algorithm candidate

is the block Wiedemann method, which computes a sequence $X^T A^i Y$ where X and Y are blocks of vectors. This step can be easily parallelized by distributing vectors of block Y to several processors. We thus have several objectives with regard to the parallelism in this paper:

- We want to solve large problems coming from homology computation.
- We want to experimentally validate our parallel implementation of the block Wiedemann rank algorithm.
- We want to show the parallel scaling of block Wiedemann approaches.

1.3 Summary of the paper

In section 2, the algorithms and optimizations we used to generate the matrices and compute with them are discussed. Then, section 3 shows our experimental results on these large sparse matrices coming from homology.

2. ALGORITHMS AND OPTIMIZATIONS

As seen in section 1.1.3, we can effectively compute the homology of a cellular space and of a group which acts "nicely" on a cellular space. The main difficulty remains to find such explicit cellular space. In section 2.1 we present the process of matrix generation and optimization. Then in the following sections we give a description of the algorithms used for the computation of the rank and the Smith form of those matrices.

2.1 Matrices generation

In the case of subgroups of $GL_N(\mathbb{Z})$, we have an "obvious" action on $\mathbb{Z}^N$. We can then capture the topology by regarding $\mathbb{Z}^N$ not as a free $\mathbb{Z}-$module but as a lattice (or equivalently as quadratic forms), and see if this leads to some interesting topological construction. We will describe this approach below and the results that we can get.

2.1.1 Voronoï's reduction theory

Let $N \geqslant 2$ be an integer. We let C_N be the set of positive definite real quadratic forms in N variables. Given $h \in C_N$, let $m(h)$ be the finite set of minimal vectors of h, i.e. vectors $v \in \mathbb{Z}^N$, $v \neq 0$, such that $h(v)$ is minimal. A form h is called *perfect* when $m(h)$ determines h up to scalar: if $h' \in C_N$ is such that $m(h') = m(h)$, then h' is proportional to h.

EXAMPLE 2.1. *The form $q(x,y) = x^2+y^2$ has minimum 1 and minimal vectors $\pm(1,0)$ and $\pm(0,1)$. Nevertheless this form is not perfect, because there is an infinite number of definite positive quadratic forms having these minimal vectors.*
On the other hand, the form $q(x,y) = x^2 + xy + y^2$ has also minimum 1 and has exactly 3 minimal vectors (up to sign), the one above and $\pm(1,-1)$. This form is perfect, the associated lattice is the "honeycomb lattice" (with optimal spheres packing in the plane), it is the only one.

Denote by C_N^* the set of non negative real quadratic forms on $\mathbb{R}^N$ the kernel of which is spanned by a proper linear subspace of $\mathbb{Q}^N$, by X_N^* the quotient of C_N^* by positive real homotheties, and by $\pi : C_N^* \to X_N^*$ the projection. Let $X_N = \pi(C_N)$ and $\partial X_N^* = X_N^* - X_N$. Let Γ be either $GL_N(\mathbb{Z})$ or $SL_N(\mathbb{Z})$. The group Γ acts on C_N^* and X_N^* on the right by the formula

$$h \cdot \gamma = \gamma^t \, h \, \gamma \,, \quad \gamma \in \Gamma \,,\ h \in C_N^* \,,$$

where h is viewed as a symmetric matrix and γ^t is the transposed of the matrix γ. Voronoï proved that there are only finitely many perfect forms modulo the action of Γ and multiplication by positive real numbers ([32], Th. p. 110).
Given $v \in \mathbb{Z}^N - \{0\}$ we let $\hat{v} \in C_N^*$ be the form defined by

$$\hat{v}(x) = (v \mid x)^2 \,,\ x \in \mathbb{R}^N \,,$$

where $(v \mid x)$ is the scalar product of v and x. The *convex hull* of a finite subset $B \subset \mathbb{Z}^N - \{0\}$ is the subset of X_N^* image by π of the elements $\sum_j \lambda_j \, \widehat{v_j}$, $v_j \in B$, $\lambda_j \geqslant 0$. For any perfect form h, we let $\sigma(h) \subset X_N^*$ be the convex hull of the set $m(h)$ of its minimal vectors. Voronoï proved in [32, § 8-15], that the cells $\sigma(h)$ and their intersections, as h runs over all perfect forms, define a cell decomposition of X_N^*, which is invariant by the action of Γ. We endow X_N^* with the corresponding CW-topology. If τ is a closed cell in X_N^* and h a perfect form with $\tau \subset \sigma(h)$, we let $m(\tau)$ be the set of vectors v in $m(h)$ such that $\hat{v}$ lies in τ. Any closed cell τ is the convex hull of $m(\tau)$ and $m(\tau) \cap m(\tau') = m(\tau \cap \tau')$.

2.1.2 Voronoï's complex

Let $d(N) = N(N+1)/2 - 1$ be the dimension of X_N^* and $n \leqslant d(N)$ a natural integer. We denote by Σ_n^* a set of representatives, modulo the action of Γ, of those cells of dimension n in X_N^* which meet X_N, and by $\Sigma_n \subset \Sigma_n^*$ the cells σ such that the stabilizer Γ_σ of σ in Γ preserves its orientation. Let V_n be the free abelian group generated by Σ_n. We define as follows a map

$$d_n : V_n \to V_{n-1} \,.$$

For each closed cell σ in X_N^* we fix an orientation of σ, i.e. an orientation of the real vector space $\mathbb{R}(\sigma)$ of symmetric matrices spanned by the forms $\hat{v}$, $v \in m(\sigma)$. Let $\sigma \in \Sigma_n$ and let τ' be a face of σ. Given a positive basis B' of $\mathbb{R}(\tau')$ we get a basis B of $\mathbb{R}(\sigma)$ by adding after B' a vector $\hat{v}$, $v \in m(\sigma) - m(\tau')$. We let $\varepsilon(\tau', \sigma) = \pm 1$ be the sign of the orientation of B in the oriented vector space $\mathbb{R}(\sigma)$ (this sign does not depend on the choice of v).

Next, let $\tau \in \Sigma_{n-1}$ be the cell equivalent to $\tau' = \tau \cdot \gamma$. We define $\eta(\tau, \tau') = 1$ (resp. $\eta(\tau, \tau') = -1$) when γ is compatible (resp. incompatible) with the chosen orientations of $\mathbb{R}(\tau)$ and $\mathbb{R}(\tau')$.

Finally we define

$$d_n(\sigma) = \sum_{\tau \in \Sigma_{n-1}} \sum_{\tau'} \eta(\tau, \tau') \, \varepsilon(\tau', \sigma) \, \tau \,, \qquad (1)$$

where τ' runs through the set of faces of σ which are equivalent to τ.
It is shown in [14], that up to p-torsions with $p \leqslant N+1$, the homology of this complex computes the cohomology of G. For $N = 5, 6, 7$ we get the following results for Σ_n.

n	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	total
$\Sigma_n^*(GL_5(\mathbb{Z}))$	5	10	16	23	25	23	16	9	4	3							**136**
$\Sigma_n(GL_5(\mathbb{Z}))$	0	0	0	1	7	6	1	0	2	3							**20**
$\Sigma_n^*(GL_6(\mathbb{Z})$	3	10	28	71	162	329	589	874	1066	1039	775	425	181	57	18	7	**5634**
$\Sigma_n(GL_6(\mathbb{Z}))$	0	0	0	0	3	46	163	340	544	636	469	200	49	5	0	0	**2455**
$\Sigma_n^*(SL_6(\mathbb{Z}))$	3	10	28	71	163	347	691	1152	1532	1551	1134	585	222	62	18	7	**7576**
$\Sigma_n(SL_6(\mathbb{Z}))$	0	3	10	18	43	169	460	815	1132	1270	970	434	114	27	14	7	**5486**

Figure 1: Cardinalities of Σ_n and Σ_n^* for $N = 5, 6$ (empty slots denote zero)

n			6	7	8	9	10	11	12	total
$\Sigma_n^*(GL_7(\mathbb{Z}))$			6	28	115	467	1882	7375	26885	**36758**
$\Sigma_n(GL_7(\mathbb{Z}))$			0	0	0	1	60	1019	8899	**9979**
n	13	14	15	16	17	18	19	20	21	total
$\Sigma_n^*(GL_7(\mathbb{Z}))$	87400	244029	569568	**1089356**	**1683368**	**2075982**	**2017914**	**1523376**	**876385**	**10167378**
$\Sigma_n(GL_7(\mathbb{Z}))$	47271	171375	460261	**955128**	**1548650**	**1955309**	**1911130**	**1437547**	**822922**	**9309593**
n			22	23	24	25	26	27	total	TOTAL
$\Sigma_n^*(GL_7(\mathbb{Z}))$			374826	115411	24623	3518	352	33	**518763**	**10722899**
$\Sigma_n(GL_7(\mathbb{Z}))$			349443	105054	21074	2798	305	33	**478707**	**9798279**

Figure 2: Cardinalities of Σ_n and Σ_n^* for $N = 7$

THEOREM 2.2. *(Elbaz-Vincent/Gangl/Soulé)[15, 13, 14]. The cardinalities of Σ_n and Σ_n^* are shown on figure 1 for $N = 5, 6$ and on figure 2 for $N = 7$.*

The previous result gives the precise size of the matrices involved in the computation of the homology.

The main challenge was then the computation of the ranks of the matrices of the differential (this gives the free part of the homology) and the computation of the Smith forms (which gives the relevant arithmetical information of the homology), in particular for $N = 7$, knowing that such matrices are particularly sparse. We can emphasize the fact that what we want to detect is the "high torsion" in the homology (i.e. the prime divisors > 7 of the Smith invariants).

In the following paragraphs we will discuss the different methods chosen for the computations and to take up the challenge[1].

2.2 Coppersmith Block Wiedemann

One successful approach to deal with linear algebra computations on large sparse matrices is to rely on Lanczos/Krylov black-box methods. In particular, block versions of Wiedemann method [33] are well suited for parallel computation. This technique has been first proposed by Coppersmith in [6] for computation over $GF(2)$ and then analyzed and proved by Kaltofen [19] and Villard [30, 31]. The idea is for a matrix $A \in \mathsf{F}^{n\times n}$ to compute the minimal generating matrix polynomial of the matrix sequence $\{XA^iY\}_{i=0}^{\infty} \in \mathsf{F}^{s\times s}$, where X, Y are blocks of s vectors (instead of vectors in the original Wiedemann's algorithm). Therefore, an intuitive parallelization of this method is to distribute the vectors of the block X, Y to several processors. Thus, the computation of the sequence, which is the major performance bottleneck of this method, can be done in parallel and then allow for better performance.

Lots of implementations and practical experimentations has been developed on parallel block Wiedemann. For instance, in 1996, Kaltofen and Lobo [20] have proposed a coarse grain implementation to solve homogeneous linear equations over $GF(2)$. They have thus been capable to solve a system of 252 252 linear equations with about 11.04 million non-zero entries, in about 26.5 hours using 4 processors of an SP-2 multiprocessor.

Lately, in 2001, Thomé in [27] improved Coppersmith's algorithm by introducing matrix half-gcd's computation, and its implementation [28] was able to outperform Kaltofen-Lobo's software. One may remark that introduction of matrix gcd was first suggested by Villard in [30] who relied on the work of Beckermann and Labahn [2] on power Hermite Padé approximation. Finally, Giorgi, Jeannerod and Villard have generalized in [16] block Wiedemann algorithms by introducing σ-basis computation and then reducing the complexity to polynomial matrix multiplication. A sequential implementation of this algorithm is now available in the LinBox library (www.linalg.org).

2.3 Block symmetry

In order to reduce the number of dot products, we used a symmetric projection. In other words, we set $X = Y^T$ in the XA^iY sequence. Indeed, in this case the probability of success is reduced but the obtained block is symmetric as

[1]All the matrices are available on line in the "Sparse Integer Matrix Collection" (`ljk.imag.fr/membres/Jean-Guillaume.Dumas/simc.html`)

soon as A is symmetric. This is always the case when the preconditioners of [10] are used (they are of the form A^TA). This reduces the dot product part of the computation of the sequence by a factor of two. For instance column i can be deduced from its top i elements and row i. This induces some load balancing issues when one process owns the computation of one column. Note also that we use BLAS level-2 for the computation of this dot products. In other words we perform them by blocks.

2.4 σ-basis computations

In order to efficiently compute σ-basis we rely on algorithm PM-Basis of [16] which reduces this computation to polynomial matrix multiplication. One can multiply two polynomial matrices $A, B \in \mathsf{F}^{n\times n}[x]$ of degree d in $O(n^3d + n^2d\log d)$ finite field operations if d-th primitive roots of unity are available in F. Consequently, we decided for our computations to define F as a prime field with primes of the form $c\times 2^k+1$ such that $c\times 2^k \equiv 0 \bmod d$. These primes are commonly called FFT primes since they allow the use of FFT evaluation/interpolation for polynomials of degree d. We refer the reader to [5, 3] and references therein for further informations on fast polynomial matrix arithmetic.

When finite fields not having d-th primitive roots of unity are used, polynomial matrix multiplication is still be done efficiently by using Chinese Remainder Theorem with few FFT primes. Let be F a prime field of cardinality p, then the multiplication of $A, B \in \mathsf{F}^{n\times n}[x]$ of degree d can be efficiently done by using CRT with FFT primes p_i satisfying $\prod p_i > d\times n\times p^2$. This is equivalent to perform the multiplication over the integers and then reduce the result in F. The overall performance of the multiplication, and then of the σ-basis , is dependent on the numbers of FFT primes needed.

2.5 Rank

Our main interest in the block Wiedemann approach is to compute the rank of large sparse matrices given by the Homology group determination problem explained in section 1.1. Hence, we rely on Kaltofen-Saunder's rank algorithm [21] and its block version [29] to achieve efficient parallel computation.

The Kaltofen-Saunders approach is based on the fact that if $\tilde{A}$ is a good preconditioned matrix of A then its rank is equal to the degree of its minimal polynomial minus its valuation (or co-degree) [21]. Thus, by using well chosen preconditioners and Wiedemann algorithm one can easily compute the rank of a sparse matrix over a finite field. The block version of this method is presented e.g. in [29, §4]. We recall now the basic outline of this algorithm.

Block Wiedemann Rank Algorithm :
let $A \in \mathsf{F}^{n\times n}$,

1. **form** $\tilde{A}$ from A with good preconditioners (e.g. those of [10]).
2. **choose** random block $Y \in \mathsf{F}^{n\times s}$ and **compute** the matrix sequence $S = \{Y^T\tilde{A}^iY\}$ for $i = 0\dots 2n/s + O(1)$.
3. **compute** the minimal matrix generator $F_Y^{\tilde{A}} \in \mathsf{F}^{s\times s}[x]$ of the matrix sequence S.
4. **return** the rank r as $r = \deg(\det(F_Y^{\tilde{A}})) - \operatorname{codeg}\det(\mathrm{F}_Y^{\tilde{A}})$.

Note that if the minimal matrix of step 3 is in Popov form (e.g. computed using the σ-basis of [16]), then the degree of $\det(F_Y^{\tilde{A}})$ is simply the sum of the row degrees of the matrix $F_Y^{\tilde{A}}$. Then the co-degree is zero if the determinant of the constant term of $F_Y^{\tilde{A}}$, seen as a matrix polynomial, is non-zero. In the latter case the computation of the determinant of the whole polynomial matrix can be avoided.

When this fails, this determinant is computed by a massively parallel evaluation/interpolation. It could be interesting, though, to interpolate only the lower coefficients of this polynomial incrementally. This was not required for the matrices we considered and we therefore did not investigate more on these speed improvements.

Note that to probabilistically compute the rank over the integers, it is sufficient to choose several primes at random and take the largest obtained value, see e.g. [9] for more details. Moreover, one can choose the primes at random among primes not dividing the determinant (and thus preserving the rank). In order to ensure this property it it sufficient to select primes not dividing the valence or last invariant factor computed by one of the methods of next section.

2.6 Smith form

The computation of the Smith form for the matrices of $GL_7(\mathbb{Z})$ turned out to be a very challenging problem.

2.6.1 Smith for via the Valence

Prior experience with sparse homology matrices led us to try the SmithViaValence algorithm of [9]. The idea is to compute the minimal valence (the coefficient of the smallest non zero monomial of the minimal polynomial) of the product A^TA to determine the primes p which divide the invariant factors of the Smith form of A. When the primes have been found, one can compute the local Smith forms of A at each p separately and return the resulting Smith form S as the product of the local Smith forms S_p over all p. Local Smith form computation can be done by a repeated Gauss elimination modulo p^e where the exponent e is adjusted automatically during the course of the algorithm.

This algorithm works very efficiently for sparse matrices provided that the minimal polynomial of the product AA^T has a small degree. Unfortunately, the latter condition does not hold in the case of $GL_7(\mathbb{Z})$ matrices. Moreover, some early experiments with small matrices showed that much more primes occur in the computed valence than in the Smith form of the original matrix.

2.6.2 Saunders and Wan's adaptive algorithm

Thus, we decided to apply the adaptive algorithm of Saunders and Wan [24] which is a modified version of Eberly-Giesbrecht-Villard algorithm [12]. In [12] the authors proposed a procedure *OneInvariantFactor*(i, A) (OIF) which computes the ith invariant factor of a $n\times n$ matrix A. Then the binary search for distinct invariant factors allows them to find the Smith form of A. OIF reduces the ith factor

computation to the computation of the last (nth) invariant factor (LIF) of a preconditioned matrix $A + U_iV_i$, where U_i, V_i^T are random $n \times (n-i)$ matrices. In [24] the method was extended to handle the rectangular case of $m \times n$ matrix. It is done by computing the last (ith) invariant factor of a preconditioned matrix L_iAR_i where L_i is a $m \times i$ and R_i is a $i \times n$ matrix.

The procedure OIF is of Monte Carlo probabilistic type where the probability of correctness is controlled by repeating the choice of preconditioners. Assuming the correctness of LIF computation, it gives a multiple of the ith invariant factor. In practice, LIF is also of randomized Monte Carlo type. The idea is to get a divisor of the last invariant factor by solving a linear equation $Mx = b$ with random right-hand side. After several solvings we get the last invariant factor with large probability. Thus, the overall situation is more complex and we cannot exclude the possibility that some primes are omitted or unnecessary in the output of OIF. However, the probability that a prime is omitted or is unnecessary in this output can be controlled for each prime separately and is smaller for bigger primes.

Therefore in [24] the authors introduce a notion of smooth and rough parts of the Smith form. The idea is to compute the local Smith form for smaller primes by for example the SmithViaValence or OIF algorithm and to recover only large primes with the invariant factor search of [12]. When we consider large primes, a sufficient probability of correctness can be obtained by a smaller number of repetitions.

2.6.3 More adaptiveness

As we did not want to compute the valence, we introduced some minor changes to the algorithm, which at the end is as follows:

1. $r = \text{rank}(A)$
2. For primes $1 < p < 100$ compute the local Smith form S_p of A;
3. Compute $s_r(A)$ by OneInvariantFactor algorithm;
4. $P =$ all primes $p > 100$ which divide $s_r(A)$;
5. If $P = \emptyset$ return $S = \prod_p S_p$;

One advantage of this method is that we get the information on the smooth form of the matrix very quickly. Moreover, the OIF computation acts as a certification phase which allows us to prove that no other primes are present with a sufficiently large probability. This probability is explicit in the following theorem:

THEOREM 2.3. *The probability that there exists a prime $p > P$ that divides the ith invariant factor but does not divide the output of OIF which uses M random preconditioners L_i, R_i and N random vectors b, with $b \in \{0, 1, \dots \beta - 1\}$ and $\beta > s_i(A)$, in the LIF procedure is bounded by*

$$M \sum_{p>P}^{\infty} \left(\frac{2}{p}\right)^N.$$

PROOF. As we take the gcd of the result with different preconditioners L_i and R_i, is suffices that the LIF computation fails in one case to spoil the computation. We are free to choose a large bound for $\|b\|$ such that $\|b\| > s_i(A)$ without increasing the complexity of LIF computation. Then the probability that a prime $p < \|b\|$ is omitted in LIF is less than or equal to $\frac{1}{\beta}\lceil\frac{\beta}{p}\rceil < \frac{2}{p}$, see [1]. Finally, we bound the probability that any prime $p > P, p \mid s_i(A)$ is missing by taking a sum over all primes. □

The choice $M = N = 2$ suffice to obtain a small probability $0,015$ of omitting an important prime, and at the same time to exclude all primes that are not in the ith invariant factor. In our experiments, there was no need to perform the computation for any additional prime $p > 100$ as all the primes were excluded by the OIF computation. This is one of the most important advantages over the valence computation.

The algorithm [24] was stated in the case of dense matrices. We slightly modified it in order to exploit the sparse structure of the matrix. In particular, we used the sparse local Smith form computation of [9, Algorithm LRE] but stick to the dense Dixon solver [7] as long as the memory was sufficient. Any other solver, including the new sparse solver of [11] could potentially be used for larger matrices.

The limits of this method are imposed by the available memory. For example, it was possible to use the dense Dixon solver only for the six smallest matrices. Furthermore, sparse elimination reached its limits for matrices of size greater than 171375×47271 and 21074×105054 when the filling of the matrices started to be impossible to handle. For the 460261×171375 matrix GL7d15 and 105054×349443 matrix GL7d23, specialized space-efficient elimination procedures mod 2, 3 and 5 allowed us to compute the rank mod 2, 3 and 5 respectively.

2.6.4 Chain Reductions

The encountered problems have shown a need for a more elaborated reduction algorithm. We focused our attention on the algebraic reduction algorithm for chain complexes of [18]. We implemented a simplified version of the algorithm in the language of matrices using the LinBox library. The heuristic behind this algorithm is that Gaussian elimination can propagate from one matrix of a chain complex to the next thanks to the exactness of the differential map (i.e. $d^2 = 0$ condition).

The motivations come from the geometric properties of homologies. By a free face we refer to a $(k-1)$ cell a which is in the differential of only one k-dimensional cell b. By removing the pair (a, b) we obtain a retract of the initial cell complex (viewed as a geometrical object), see **Figure** 3. The process can be repeated. From the homology theory we know that the groups of homology are the same for the set and its retract. The removal of pairs leads to a reduction of the basis of the cell complex. We refer to [17, Ch.4] for a full description of the procedure.

If the differential map is represented by matrices whose rows represent the cells of dimension $k-1$ and columns - dimension k, the removal of a pair (a, b) can be interpreted as the

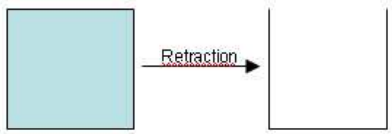


Figure 3: Retraction for a square

removal of a row with only one non-zero entry (1-row) and the column it points to. In the general case, the algebraic reduction of (a, b) such that $d_k(b) = \lambda a + u$ is possible iff λ is invertible in the ring of computation i.e. $\mathbb{Q}, \mathbb{Z}, \mathbb{Z}_{p^k}$ - depending on the problem. A modification $\tilde{d}$ of the differential given by the formula

$$\tilde{d}_k v = d_k(v) - \lambda^{-1}[v : a]d_k(b). \quad (2)$$

Thus, in the basic case of free face removal no modification is needed. In the case of matrices, the formula describes a step of Gauss elimination where the reduced row is removed and not permuted. This proves that the Smith form (or the rank) of the initial and reduced matrix will be the same, provided we add a number of trivial invariant factors equal to the number of rows reduced to the reduced Smith form.

The important characteristic of this methods is that, thanks to the exactness of the matrix sequence, we can also remove row b and column a from the neighboring matrices. In this way, elimination in one matrix can propagate on the others.

Due to the format of data (large files with matrix entries) we decided to implement only the simplest case of 1-rows removal which led to entries removal but no modifications. We removed the empty rows/columns at the same time and performed the whole reduction phase at the moment of reading the files. This led to vast matrix reduction in the case of $GL_7(\mathbb{Z})$ matrices from the beginning of the sequence. The propagation of reductions unfortunately burned out near $GL7d14$ matrix and stopped completely on $GL7d19$. Applying the process for the transposed sequence did not improve the solution. Next step would be to implement the propagation of Gauss elimination steps as in Eq. (2). It would be interesting to examine whether the burn-out can be connected to the loss of regularity for X/G and/or a huge rectangularity of the input matrix $GL7d14$.

3. EXPERIMENTAL RESULTS

3.1 Parallel implementation

For the sake of simplicity in our experimental validation, we decided to develop our parallel implementation using shell tasks distribution on SMP architecture. Thus, a simple script code is used to distribute all different tasks over all the processors and files are used to gather up the computed results. Our parallel implementation has been done as follow:

1. The block of vectors Y and the sparse matrix A are broadcasted on every processors.
2. Each process takes one column of the blackbox $A^T.A$ and compute the corresponding column's sequence using the first ith columns of Y. Each process writes the result in a file labeled with the corresponding index of the column sequence.
3. When all previous processes have terminated, the σ-basis computation is sequentially performed after loading the sequence from the generated files.

Despite the naïve approach used for the parallelization, our implementation authorized us to perform very large computations as show in next section. However, our experiments show a need for at least a more robust parallel computation scheduler.

3.2 Rank and Smith form

All our computation have been done on a SGI Altix 3700 gathering 64 Itanium2 processors with 192Gb memory and running SuSE Linux Enterprise System 10. Further informations on this platform are available at `http://www.math.uwaterloo.ca/mf`

In **Table** 1 we include the information about the dimensions of the $GL_7(\mathbb{Z})$ matrices and their sparsity. The matrices are very sparse which is illustrated by the fact that less than 1% of the entries are non-zero except for matrices GL7d10 and GL7d11. This value drops to less than 0,2% in the case of the largest matrices. Also in **Table** 1 we give the results for the rank and the Smith form computations. We have obtained a full information on the rank of $GL_7(\mathbb{Z})$ matrices. For the computation of the Smith form, full result has been obtained in the case of matrices 10,11,12,13,25,26. For matrices 14 and 24 only the smooth part of the Smith form has been computed. For matrices 15 and 23 we have proved the existence of a non-trivial local Smith form at 2 and 3 and a triviality of the local Smith form at 5. As the result for these matrices we give the number of invariant factors divisible by 2 and 3.

In **Table** 2 we give the times for the Smith form algorithms used. For cases with * no data are available or relevant.

The rank computation for $GL_7(\mathbb{Z})$ matrices was performed modulo 65537. This FFT prime allowed us to use both BLAS routines and sigma-basis reconstruction using fast polynomial multiplication. In **Table** 3 we give the timings for different operations used in the computation i.e. the sparse matrix-vector product, BLAS-based matrix-vector product and, for the sake of comparison, the time of scalar dot product equivalent to the BLAS computation.

In **Table** 4 we give the estimation of sequential and parallel cpu time of rank computation and compare it with the real time of parallel computation. The times are estimated based on the number of iterations and the times of one step which can be computed from **Table** 3 (notice, that in the scalar case we use 1 dot product instead of BLAS). The real time of computation includes the time of writing and reading the data which was considerable. The difference of the real and estimated running times may also be due to the overload of the computation cluster. Moreover, long computations suffered from system crashes and/or shutdowns. Some restoration scripts were used to recover the data which

A	Ω	n	m	rank	ker	Smith form
GL7d10	8	60	1	1	59	1
GL7d11	1513	1019	60	59	960	1 (59)
GL7d12	37519	8899	1019	960	7939	1 (958), 2 (2)
GL7d13	356232	47271	8899	7938	39333	1 (7937), 2 (1)
GL7d14	1831183	171375	47271	39332	132043	1 (39300),2 (29),4 (3)
GL7d15	6080381	460261	171375	132043	28218	1 (131993), 2·? (46), 6·? (4) (*)
GL7d16	14488881	955128	460261	328218	626910	
GL7d17	25978098	1548650	955128	626910	921740	
GL7d18	35590540	1955309	1548650	921740	1033569	
GL7d19	37322725	1911130	1955309	1033568	877562	
GL7d20	29893084	1437547	1911130	877562	559985	
GL7d21	18174775	822922	1437547	559985	262937	
GL7d22	8251000	349443	822922	262937	86506	
GL7d23	2695430	105054	349443	86505	18549	1 (86488), 2·? (12), 6·? (5) (*)
GL7d24	593892	21074	105054	18549	2525	1 (18544),2 (4),4 (1)
GL7d25	81671	2798	21074	2525	273	1 (2507), 2 (18)
GL7d26	7412	305	2798	273	32	1 (258), 2 (7), 6 (7), 36 (1)

Table 1: Results of the rank and Smith form computation for $GL_7(\mathbb{Z})$ matrix A of dimension $n \times m$ with Ω non-zero entries. For (*) the information is incomplete - only divisors of the invariant factors were determined based on the rank mod 2 and 3 computation.

A	$\tilde{n}$	$\tilde{m}$	$\tilde{r}$	Red	RAdaptive	SmoothSF	AdaptiveSF	SFValence
GL7d11	39	8	52	0.01s	$< 10^{-2}$s	0.09s	0.26s	4.84s
GL7d12	289	58	909	0.30s	0.16s	9.75s	218.68s	4.04h
GL7d13	7938	740	7250	3.12s	159.16s	0.76h	*	2526.65h
GL7d14	165450	35741	4279	21.62s	*	796h	*	*
GL7d25	2797	20990	0	1.74s	*	17.67s	4.40h	52.13h
GL7d26	302	2748	0	0.14s	*	0.29s	26.81s	274.35s

Table 2: Times for Smith Form computation for $GL_7(\mathbb{Z})$ matrices. From left to right: the dimensions of the matrix after reductions, rank approximation by reductions, time of reading and reducing the matrix, time for the adaptive algorithm for a reduced matrix; times for the original matrix: smooth form computation, adaptive algorithm; valence computation in parallel - sequential time equivalent.

A	iter [1]	time app	iter [p]	time	time app	σ-basis
GL7d11	120	0.02s	6 [30]	0.01s	$< 10^{-2}$s	0.58s
GL7d12	1922	7.53s	66 [30]	0.32s	0.26s	12.16s
GL7d13	15878	880.28s	532 [30]	51.65s	29.49s	249.17s
GL7d14	78666	5.90h	2625 [30]	0.56h	0.20h	0.45h
GL7d15	264088	66.98h	8805 [30]	2.25h	2.23h	2.45h
GL7d16	656438	509.80h	21884 [30]	27.29h	17.00h	6.03h
GL7d17	1253822	113.90d	41796 [30]	14d	3.80d	0.57d
GL7d18	1843482	256.50d	46089 [40]	28d	6.41d	1.00d
GL7d19	2067138	321.89d	41345 [50]	35d	6.44d	1.56d
GL7d20	1755126	212.82d	36568 [48]	10d	4.43d	1.41d
GL7d21	1119972	75.01d	37335 [30]	5d	2.50d	0.55d
GL7d22	525876	293.60h	17532 [30]	16.47h	9.79h	5.85h
GL7d23	173012	21.18h	5769 [30]	1.17h	0.71h	1.09h
GL7d24	37100	3172.79s	1239 [30]	188.78s	105.96s	666.83s
GL7d25	5052	40.21s	171 [30]	1.56s	1.36s	41.47s
GL7d26	548	0.40s	21 [30]	0.03s	0.02s	2.03s

Table 4: A summary of large-scale parallel rank computation. From left to right: number of iteration in the scalar case, time estimation in this case, number of iterations on p processors computed as $2 + 2 \cdot r/p$, average (real) time of sequence computation, estimated time on p processors, the time of the σ basis computation for a sequence of length iter $[p]$ of $p \times p$ matrices.

A	$1A^TAu$ [s]	$1U^Tv$ [s]	$30u^Tv$ [s]
GL7d11	0.0002	$< 10^{-4}$	$< 10^{-4}$
GL7d12	0.0038	0.0001	0.0002
GL7d13	0.0550	0.0005	0.0036
GL7d14	0.2677	0.0025	0.0190
GL7d15	0.9048	0.0082	0.0708
GL7d16	2.7724	0.0234	0.2641
GL7d17	7.8003	0.0485	0.5052
GL7d18	11.9457	0.0759	0.8710
GL7d19	13.3591	0.0948	1.0710
GL7d20	10.4056	0.0711	0.8587
GL7d21	5.7461	0.0408	0.4604
GL7d22	1.9919	0.0180	0.2082
GL7d23	0.4354	0.0052	0.0459
GL7d24	0.0843	0.0012	0.0085
GL7d25	0.0078	0.0002	0.0008
GL7d26	0.0007	$< 10^{-4}$	$< 10^{-4}$

Table 3: CPU timings (in sec.) for different operations used in large-scale parallel rank computation. All times in seconds. From left to right: time of a matrix-vector product, a BLAS multiplication of a vector and a $30 \times \min(n, m)$ matrix U and 30 dot products.

unfortunately required re-running some part of the computation. Thus, the real time given in **Table** 4[Col.5] should be treated as a rough approximation.

4. CONCLUSION

Using the previous methods and computations, we get the following new result for the rational cohomology of $GL_7(\mathbb{Z})$.

THEOREM 4.1. *(Elbaz-Vincent/Gangl/Soulé)[14] We have*

$$H^m(GL_7(\mathbb{Z}), \mathbb{Q}) = \begin{cases} \mathbb{Q} & \text{if} \quad m = 0, 5, 11, 14, 15, \\ 0 & \text{otherwise.} \end{cases}$$

Clearly the simple parallelization we used together with the highly optimized routines were the key to enable these computations.

To go further and solve even larger problems, it is mandatory to improve the parallelism. On SMP we can split the matrices into blocks and perform the matrix-vector products with different threads. This, and the unbalanced load we had when we choose to assign one vector to one process, advocates for the use of more advanced scheduling. We are experimenting KAAPI[2] but were not ready for the computation of GL_7.

Other improvements are of algorithmic type. They include the use of the sparse projections of [11] for the matrix sequence. But then we loose the symmetry of the projections and therefore must pay a factor of two for the number of iterations. We could also use an early termination strategy to stop the iteration earlier, but up to now this require to loose the fast algorithm for the sigma bases. Then if a good structure for the sparsity of the matrices could be found, e.g. an adapted reordering technique, this would enable an efficient clustering and therefore faster and more scalable matrix-vector products.

In order to have the relevant part of the torsion of the integral cohomology of $GL_7(\mathbb{Z})$, we would need the complete description of the Smith forms of all the matrices described above. Our experiments have shown that this can be an enormously difficult task. The computation remains to be done but would have applications in number theory.

[2]Kernel for Adaptive, Asynchronous Parallel and Interactive programming, `kaapi.gforge.inria.fr`

5. ACKNOWLEDGMENTS

We are grateful to Arne Storjohann and to the Computer Science Computing Facilities of the University of Waterloo for letting us fill up their SMP machine to perform our parallel computations.

6. REFERENCES

[1] J. Abbott, M. Bronstein, and T. Mulders. Fast deterministic computation of determinants of dense matrices. In S. Dooley, editor, *Proceedings of the 1999 International Symposium on Symbolic and Algebraic Computation, Vancouver, Canada*, pages 197–204. ACM Press, New York, July 1999.

[2] B. Beckermann and G. Labahn. A uniform approach for the fast computation of matrix-type Padé approximants. *SIAM Journal on Matrix Analysis and Applications*, 15(3):804–823, 1994.

[3] A. Bostan and E. Schost. Polynomial evaluation and interpolation on special sets of points. *J. Complex.*, 21(4):420–446, 2005.

[4] K. S. Brown. *Cohomology of groups.* Graduate Texts in Mathematics, 87. New York-Heidelberg-Berlin: Springer- Verlag. X, 306 p., 4 figs. DM 74.00 $ 29.60 , 1982.

[5] D. G. Cantor and E. Kaltofen. On fast multiplication of polynomials over arbitrary algebras. *Acta Inf.*, 28(7):693–701, 1991.

[6] D. Coppersmith. Solving homogeneous linear equations over GF[2] via block Wiedemann algorithm. *Mathematics of Computation*, 62(205):333–350, Jan. 1994.

[7] J. D. Dixon. Exact solution of linear equations using p-adic expansions. *Numerische Mathematik*, 40(1):137–141, Feb. 1982.

[8] J.-G. Dumas, editor. *ISSAC'2006. Proceedings of the 2006 International Symposium on Symbolic and Algebraic Computation, Santander, Spain.* ACM Press, New York, July 2006.

[9] J.-G. Dumas, B. D. Saunders, and G. Villard. On efficient sparse integer matrix Smith normal form computations. *Journal of Symbolic Computations*, 32(1/2):71–99, jul–aug 2001.

[10] J.-G. Dumas and G. Villard. Computing the rank of sparse matrices over finite fields. In V. G. Ganzha, E. W. Mayr, and E. V. Vorozhtsov, editors,

Proceedings of the fifth International Workshop on Computer Algebra in Scientific Computing, Yalta, Ukraine, pages 47–62. Technische Universität München, Germany, Sept. 2002.

[11] W. Eberly, M. Giesbrecht, P. Giorgi, A. Storjohann, and G. Villard. Solving sparse rational linear systems. In Dumas [8], pages 63–70.

[12] W. Eberly, M. Giesbrecht, and G. Villard. On computing the determinant and Smith form of an integer matrix. In *Proceedings of the 41st Annual Symposium on Foundations of Computer Science*, pages 675–687. IEEE Computer Society, 2000.

[13] P. Elbaz-Vincent. Perfects lattices, homology of modular groups and algebraic k-theory. *Oberwolfach Reports (OWR)*, 2, 2005. based on joint work with H. Gangl and C. Soulé.

[14] P. Elbaz-Vincent, H. Gangl, and C. Soulé. Perfect forms, cohomology of modular groups and k-theory of integers. in preparation.

[15] P. Elbaz-Vincent, H. Gangl, and C. Soulé. Quelques calculs de la cohomologie de $GL_N(\mathbb{Z})$ et de la k-théorie de $\mathbb{Z}$. *C. R. Acad. Sci. Paris, Ser. I*, 335:321–324, 2002.

[16] P. Giorgi, C.-P. Jeannerod, and G. Villard. On the complexity of polynomial matrix computations. In R. Sendra, editor, *Proceedings of the 2003 International Symposium on Symbolic and Algebraic Computation, Philadelphia, Pennsylvania, USA*, pages 135–142. ACM Press, New York, Aug. 2003.

[17] T. Kaczyński, K. Mischaikow, and M. Mrozek. *Computational Homology*. Springer, 2004.

[18] T. Kaczyński, M. Mrozek, and M. Ślusarek. Homology computation by reduction of chain complexes. *Computers and Mathematics*, 35(4):59–70, 1998.

[19] E. Kaltofen. Analysis of Coppersmith's block Wiedemann algorithm for the parallel solution of sparse linear systems. *Mathematics of Computation*, 64(210):777–806, Apr. 1995.

[20] E. Kaltofen and A. Lobo. Distributed matrix-free solution of large sparse linear systems over finite fields. In A. Tentner, editor, *Proceedings of High Performance Computing 1996, San Diego, California*. Society for Computer Simulation, Simulation Councils, Inc., Apr. 1996.

[21] E. Kaltofen and B. D. Saunders. On Wiedemann's method of solving sparse linear systems. In *Applied Algebra, Algebraic Algorithms and Error–Correcting Codes (AAECC '91)*, volume 539 of *Lecture Notes in Computer Science*, pages 29–38, Oct. 1991.

[22] M. Kurihara. Some remarks on conjectures about cyclotomic fields and K-groups of $\mathbb{Z}$. *Compos. Math.*, 81(2):223–236, 1992.

[23] J. Rosenberg. *Algebraic K-Theory and its applications*. Springer, 1995.

[24] B. D. Saunders and Z. Wan. Smith normal form of dense integer matrices, fast algorithms into practice. In J. Gutierrez, editor, *Proceedings of the 2004 International Symposium on Symbolic and Algebraic Computation, Santander, Spain*, pages 274–281. ACM Press, New York, July 2004.

[25] C. Soulé. Perfects forms and the vandiver conjecture. *J. reine angew. Math.*, 517:209–221, 1999.

[26] E. H. Spanier. *Algebraic Topology*. Springer, 1994.

[27] E. Thomé. Fast computation of linear generators for matrix sequences and application to the block Wiedemann algorithm. In *International Symposium on Symbolic and Algebraic Computation, London, Ontario*, pages 323–331. ACM Press, July 2001.

[28] E. Thomé. Subquadratic computation of vector generating polynomials and improvement of the block Wiedemann algorithm. *Journal of Symbolic Computations*, 33(5):757–775, July 2002.

[29] W. J. Turner. A block wiedemann rank algorithm. In Dumas [8], pages 332–339.

[30] G. Villard. Further analysis of Coppersmith's block Wiedemann algorithm for the solution of sparse linear systems. In W. W. Küchlin, editor, *Proceedings of the 1997 International Symposium on Symbolic and Algebraic Computation, Maui, Hawaii*, pages 32–39. ACM Press, New York, July 1997.

[31] G. Villard. A study of Coppersmith's block Wiedemann algorithm using matrix polynomials. Technical Report 975–IM, LMC/IMAG, Apr. 1997.

[32] G. Voronoi. Nouvelles applications des paramètres continus à la théorie des formes quadratiques i. *J. Crelle*, 133:97–178, 1907.

[33] D. H. Wiedemann. Solving sparse linear equations over finite fields. *IEEE Transactions on Information Theory*, 32(1):54–62, Jan. 1986.